# A Survey of Open Source Test Definitions

Tim Bird

Fuego Test System Maintainer

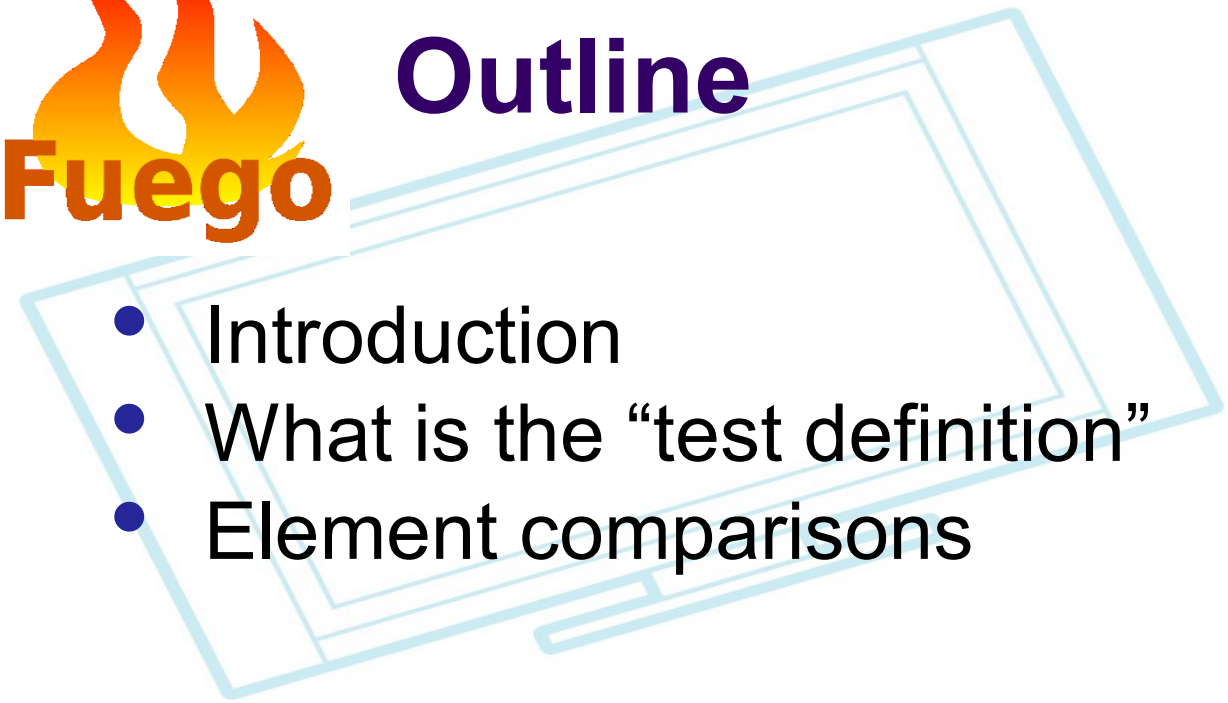
Sr. Staff Software Engineer, Sony Electronics


# Abstract

Different test frameworks have commonalities and differences in the elements of their test definitions. In this presentation, Tim will present the results of his research into different systems' test definitions, with the goal of coming up with those canonical fields which could be used for test interchange between systems. Last year, Tim conducted a survey of different test systems, and recorded the results on the elinux wiki. Combined with research this year, Tim will show the different categories of elements, describe the intersection of elements between systems, and describe difficulties in harmonizing the elements for a single unified test definition.

# Outline

- Introduction
- What is the “test definition”
- Element comparisons

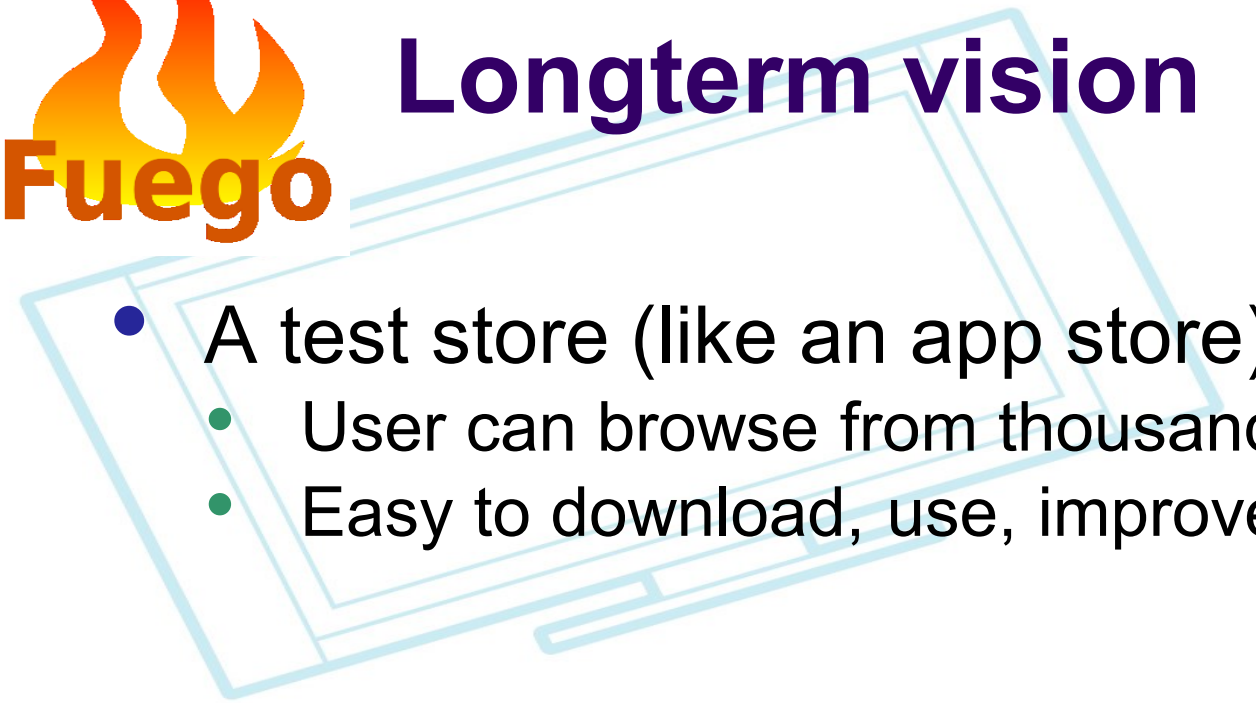


# Reason for Survey

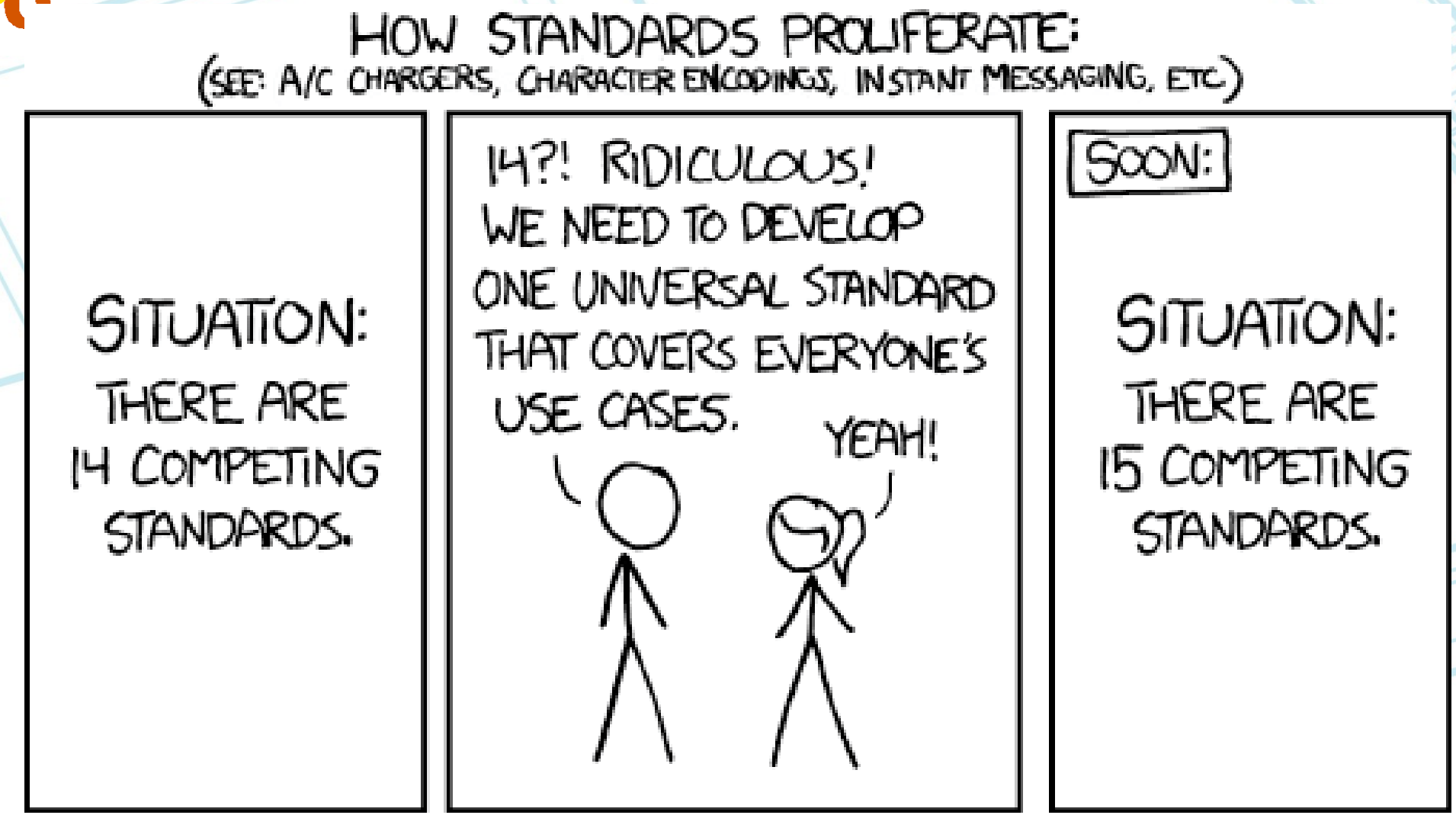
- Examine different test systems
  - See what fields each system uses
  - Find commonalities
  - Find unique ideas (outliers)
- Propose a standard schema
- So that we can interchange tests
  - Every framework can use tests from LAVA, Linaro, Fuego, 0day, syzbot, LTP, kselftest
  - Don’t want to collapse ecosystems, but expand options for users
  - Improve capabilities of tests
    - More users = more test feedback

# Longterm vision

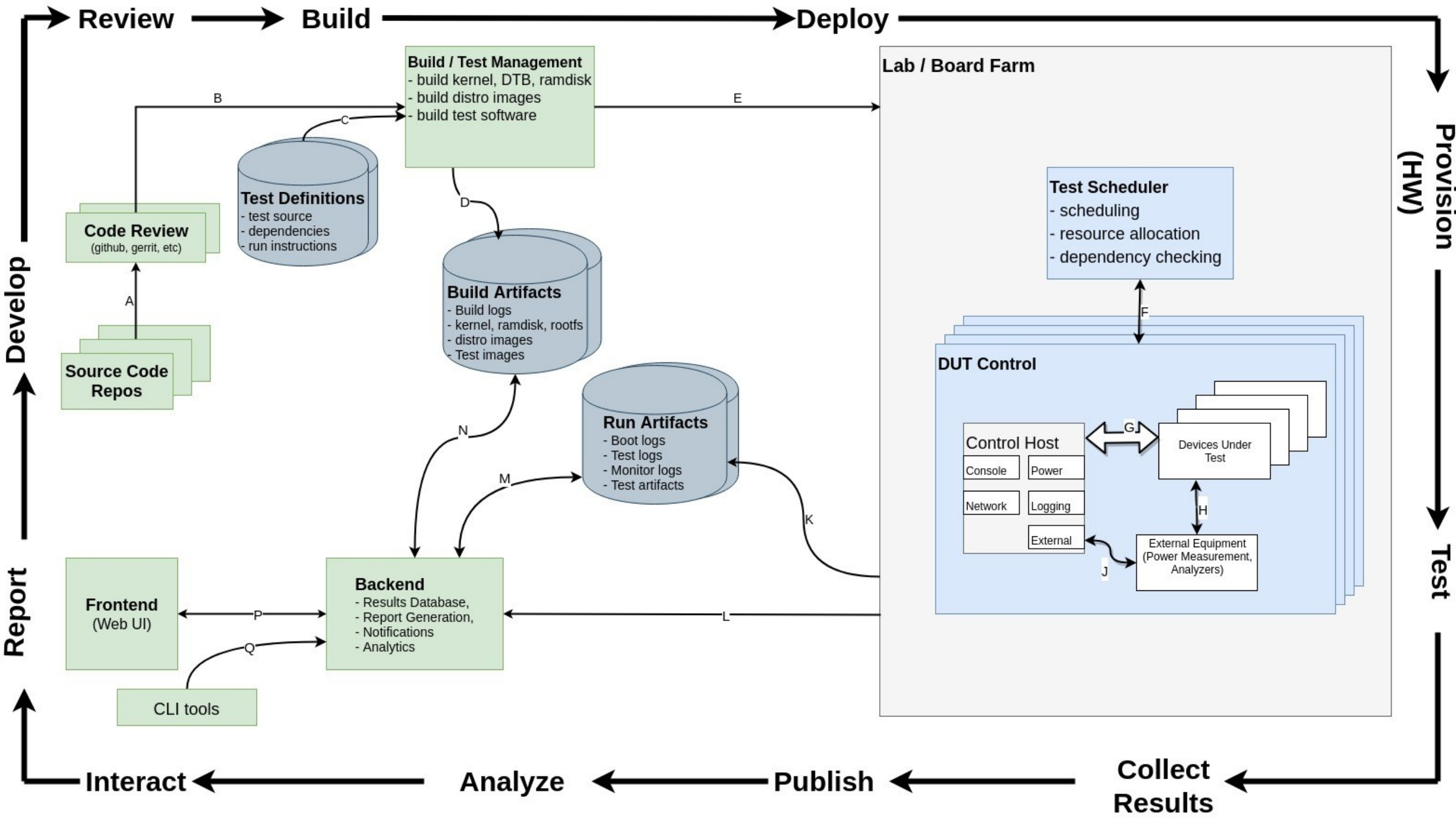
- A test store (like an app store)
  - User can browse from thousands of tests
  - Easy to download, use, improve and contribute to tests



# But, a caution about Standards...




Source: https://xkcd.com/927/ License: see https://xkcd.com/license.html

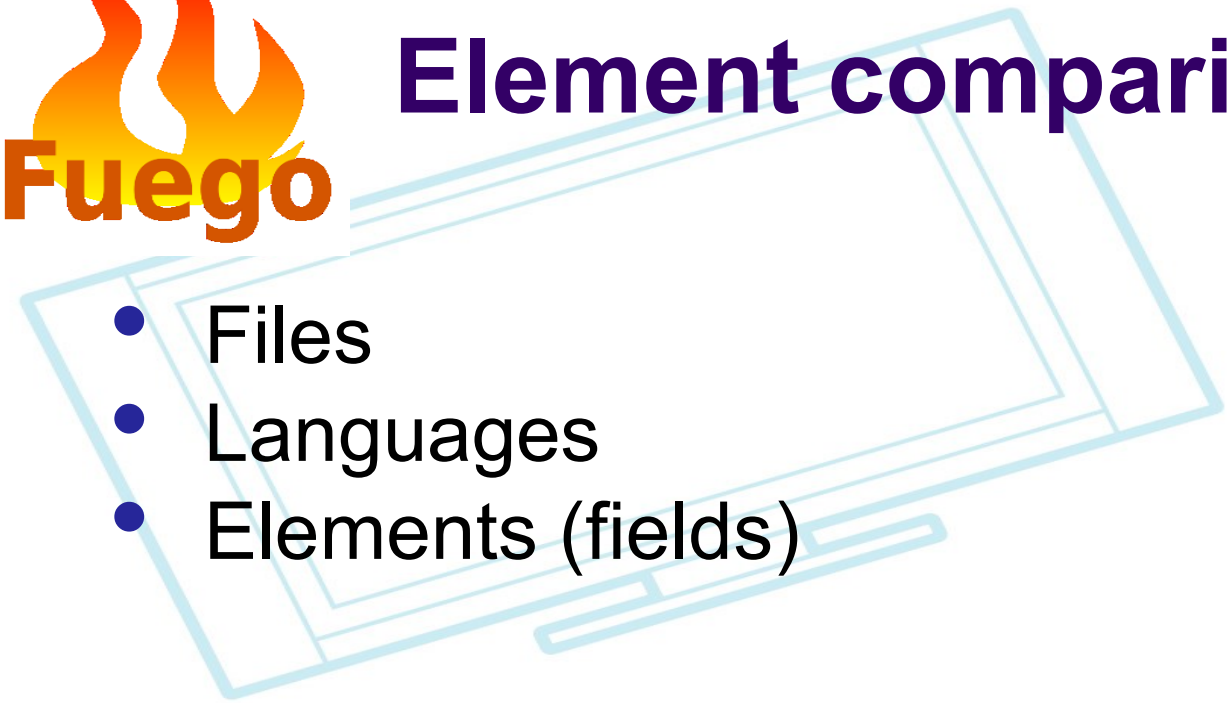

Review
Build
Deploy
Provision (HW)
Test
Collect Results
Publish
Analyze
Interact
Report
Develop
Build / Test Management
- build kernel, DTB, ramdisk
- build distro images
- build test software
Test Definitions
- test source
- dependencies
- run instructions
Code Review
(github, gerrit, etc)
Source Code Repos
Build Artifacts
- Build logs
- kernel, ramdisk, rootfs
- distro images
- Test images
Run Artifacts
- Boot logs
- Test logs
- Monitor logs
- Test artifacts
Lab / Board Farm
Test Scheduler
- scheduling
- resource allocation
- dependency checking
DUT Control
Control Host
Console
Power
Network
Logging
External
Devices Under Test
External Equipment (Power Measurement, Analyzers)
Backend
- Results Database,
- Report Generation,
- Notifications
- Analytics
Frontend
(Web UI)
CLI tools
A
B
C
D
E
F
G
H
J
K
L
M
N
P
Q

# What is a “Test definition”?

- Meta-data and instructions for a running a test
- Elements:
  - Information about a test
  - Execution control
    - Pre-requisites and dependencies
    - Test variables (parameters)
  - Instructions for test execution
  - Output parsing
  - Results format
  - Results analysis
  - Visualization control
- More practically: it’s whatever you keep in your test database

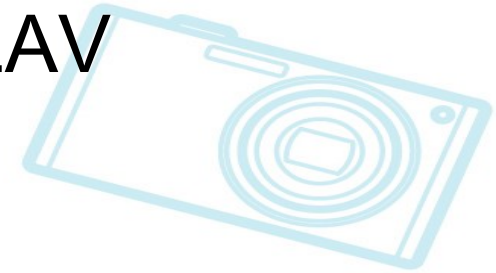
# Element comparisons

- Files
- Languages
- Elements (fields)



# Systems compared

- Fuego
- Linaro/Lava
- 0day
- Yocto Project
- CKI
- Jenkins
- SLAV

# Files - Fuego

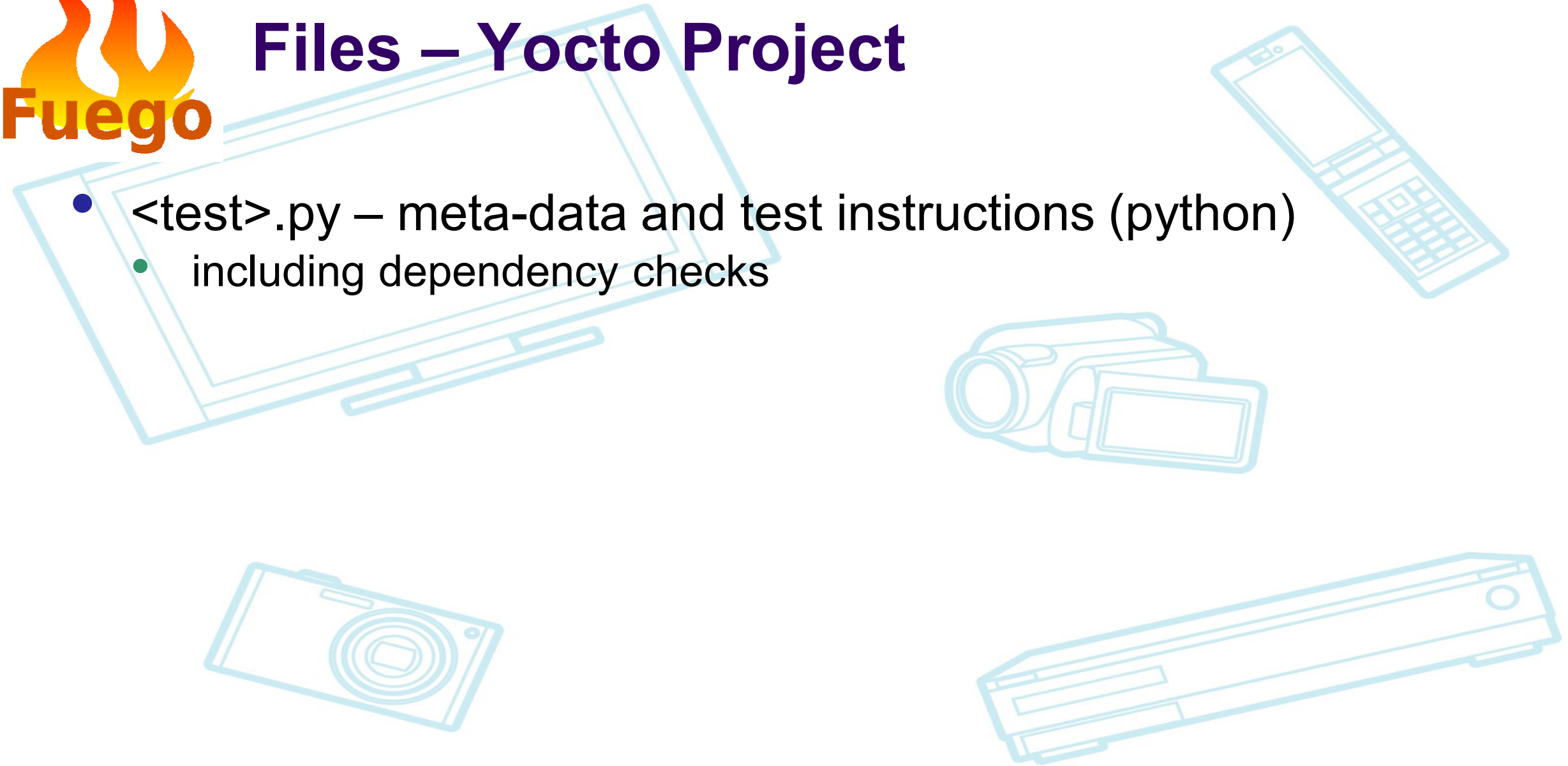
- Fuego
  - test.yaml – meta-data (test info)
  - <tarfile>-<version>.tar.gz – source for test program
  - spec.json – test variable definitions
  - fuego_test.sh – pre_requisites, test instructions
  - parser.py – parser
  - criteria.json – results analysis
  - chart_config.json – results formatting

# Files – Linaro/LAVA

- <test>.yaml – test meta-data
- <test>.sh – test instructions
  - including dependency checks

# Files – Yocto Project

- <test>.py – meta-data and test instructions (python)
  - including dependency checks



# Files – 0day

- pkg/<test>/PKGBUILD – test meta-data
  - Derived from Arch Linux package file format
- pkg/<test>/<test>.install – install instructions (in sh format)
- tests/<test> - test instructions (in sh format)
- pack/<test> - build and install instructions (in bash format)
- jobs/<test>.yaml – meta-data for job execution, including
  - Dependencies, information, job control, parameters
- Rare ones:
  - pkg/<test>/<test><version>.sig – signature of source tar

# Files - CKI

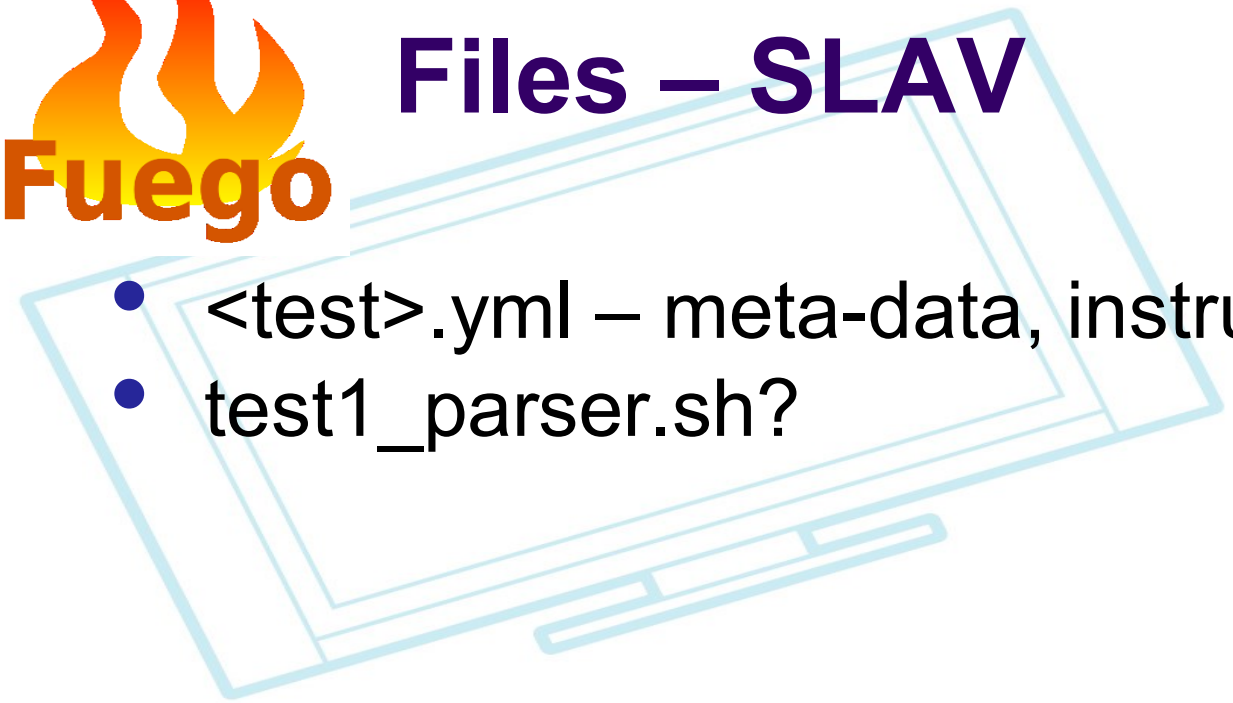
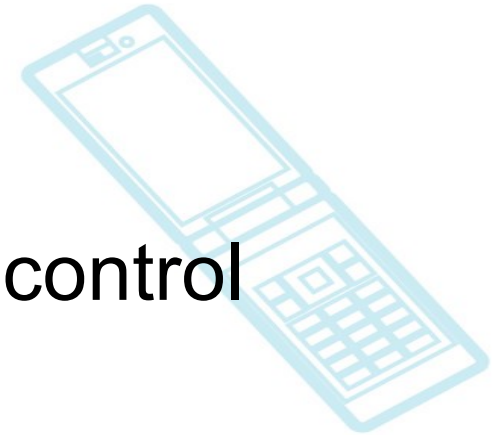
- <test>-index – meta-data, triggers control, scheduling data
  - test variables (environment)
  - testcase definitions
- Makefile – test phases (targets ‘run’, ‘build’, ‘clean’)
- metadata - test meta-data
  - Embedded in or created from Makefile
- runtest.sh – test instructions
- README.md – meta-data (markdown)
- PURPOSE – meta-data (text)

# Files – Jenkins

- jobs/<test>/config.xml – meta-data, including instructions

# Files – SLAV

- <test>.yml – meta-data, instructions, execution control
- test1_parser.sh?


# File types/Languages summary

| File type | Description | Fuego | Linaro /LAVA | Yocto | 0day | CKI | Jenkins | SLAV |
|---|---|---|---|---|---|---|---|---|
| shell script or shell snippet | Shell variables, lines or functions | yes | yes | no | yes | yes | yes | no |
| yaml | Yet another markup language | yes | yes | no | yes | no | no | yes |
| json | Javascript object notation | yes | no | rarely | no | no | no | no |
| python | Python code or fragments | yes | rarely | yes | no | rarely | no | no |
| custom | custom file format or language | no | no | no | no | yes | yes[1] | no |
| other | some other file format or language | no | no | no | ruby | make | yes[1] | yes[2] |

[1] *Jenkins has: xml, pipeline language, groovy*

[2] *SLAV has: go, haskell*


# File types/Languages summary

| File type | Description | Fuego | Linaro /LAVA | Yocto | 0day | CKI | Jenkins | SLAV |
|---|---|---|---|---|---|---|---|---|
| shell script or shell snippet | Shell variables, lines or functions | yes | yes | no | yes | yes | yes | no |
| yaml | Yet another markup language | yes | yes | no | yes | no | no | yes |
| json | Javascript object notation | yes | no | rarely | no | no | no | no |
| python | Python code or fragments | yes | rarely | yes | no | rarely | no | no |
| custom | custom file format or language | no | no | no | no | yes | yes[1] | no |
| other | some other file format or language | no | no | no | ruby | make | yes[1] | yes[2] |

[1] *Jenkins has: xml, pipeline language, groovy*

[2] *SLAV has: go, haskell*

# File types/Languages summary

| File type | Description | Fuego | Linaro /LAVA | Yocto | 0day | CKI | Jenkins | SLAV |
|---|---|---|---|---|---|---|---|---|
| shell script or shell snippet | Shell variables, lines or functions | yes | yes | no | yes | yes | yes | no |
| yaml | Yet another markup language | yes | yes | no | yes | no | no | yes |
| json | Javascript object notation | yes | no | rarely | no | no | no | no |
| python | Python code or fragments | yes | rarely | yes | no | rarely | no | no |
| custom | custom file format or language | no | no | no | no | yes | yes[1] | no |
| other | some other file format or language | no | no | no | ruby | make | yes[1] | yes[2] |

[1] *Jenkins has: xml, pipeline language, groovy*

[2] *SLAV has: go, haskell*

# Elements Comparison

- Element categories:
  - Information about a test (meta-data)
  - Provisioning
  - Execution control
    - Pre-requisites and dependencies
    - Timeouts
    - Test variables
  - Instructions for test execution
  - Output parsing or conversion
  - Results format
  - Results analysis
  - Report and notification data
  - Visualization control

# Elements - Fuego

- fuego_package_version
- name
- description
- license
- author
- maintainer
- version
- fuego_release
- type
- tags
- tarball_src
- gitrepo
- params

- data_files
- tarball
- NEED_KCONFIG
- NEED_MODULE
- NEED_ROOT
- NEED_PROGRAM
- NEED_MEMORY
- test_cleanup()
- test_build()
- test_pre_check()
- test_processing()
- test_run()
- specs

- <spec variable>
- criteria
- chart_config
- timeout
- reboot
- rebuild
- precleanup
- postcleanup

# Element Categories - Fuego

- **Info**: fuego_package_version, name, description, license, author, maintainer, version, fuego_release, tarball_src, type, tags, params
- **Execution control**: tarball, gitrepo, timeout, reboot, rebuild, precleanup, postcleanup, specs, <spec variables>, data_files
- **Pre-requisites**: NEED_KCONFIG, NEED_MODULE, NEED_ROOT, NEED_PROGRAM, NEED_MEMORY
- **Test instructions**: test_cleanup(), test_build(), test_pre_check(), test_processing(), test_run()
- **Output parsing**: parser.py
- **Results analysis**: criteria.json
- **Visualization control**: chart_config.json

# Elements - Linaro

- name
- description
- maintainer
- params
- format
- os
- scope
- devices
- run:steps

- environment
- (test instructions)
- check_root()
- install_deps()

# Element categories - Linaro

- **Info**: name, description, maintainer
- **Execution control**: params, format, os, scope, devices, environment
- **Dependencies**: inside <test>.sh, check_root(), install_deps()
- **Test instructions**: run:steps, <test>.sh
- **Output conversion**:
  - *code inside <test>.sh*

# Elements – 0day

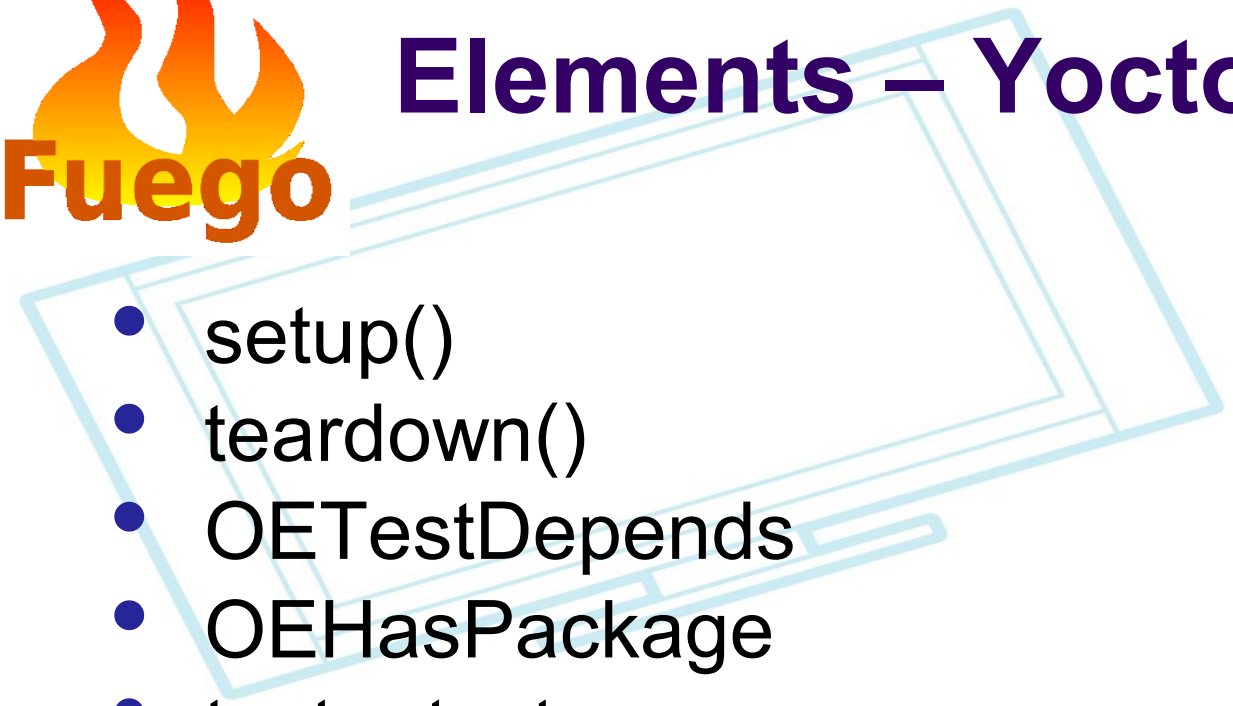
- validpgpkeys
- pkgname
- pkgver
- pkgrel
- pkgdesc
- arch
- url
- license
- depends
- optdepends
- install
- source
- sha512sums
- build()
- package()
- md5sums
- tar.sig
- post_install()
- post_upgrade()
- (test instructions)
- filename
- WEB_URL
- download()
- build()
- install()
- pack()
- suite
- testcase
- category
- disk
- fs
- iosched
- disk
- runtime
- nr_task
- disable_latency_stats
- nr_threads
- cluster
- need_kconfig
- need_memory
- need_x
- need_kernel_headers
- need_kernel_selftests
- need_cpu
- (parser)

# Element categories – 0day

- **Info**: pkgname, pkgver, pkgrel, pkgdesc, arch, url, license, sha512sums, md5sum, filename, WEB_URL, suite, category, validpkgkeys
- **Execution control**: source, disk, fs, iosched, runtime, nr_task, disable_latency_stats, nr_threads, cluster, tar_sig
- **Dependencies**: depends, optdepends, need_kconfig, need_memory, need_x, need_kernel_headers, need_kernel_selftests, need_cpu
- **Test instructions**: download(), build(), package(), install() package(), post_install(), post_upgrade(), (sh test instructions)
- **Output parsing**: (ruby parser)

# Elements – Yocto Project

- setup()
- teardown()
- OETestDepends
- OEHasPackage
- test_<testcase_name>



# Element categories – Yocto Project

- **Test Instructions**:
  - setup(), teardown(), test_<testcase_name>()
- **Dependencies**:
  - OETestDepends, OEHasPackage

# Elements - CKI

- TEST
- TESTVERSION
- BUILT_FILES
- FILES
- run:
- runtest:
- build:
- clean:
- Owner
- name
- Path
- Description
- Architecture
- Type
- TestTime
- TestVersion
- RunFor
- Priority
- License
- Confidential
- Destructive
- Requires
- Environment
- Bug
- Releases
- RepoRequires
- RhtsRequires
- startup()
- cleanup()
- runtest()
- PURPOSE
- description
- Test Maintainer
- url_suffix
- pattern
- sets
- trees
- sources
- arches
- components
- cases
- name
- environment
- host_type_regex
- hostRequires
- kickstart
- waived
- role
- max_duration_seconds

# Element categories - CKI

- **Info**: TEST, TESTVERSION, BUILT_FILES, FILES, Owner, name, Path, Description, Architecture, Type, TestVersion, Bug, PURPOSE, Test Maintainer, cases, cases/description, cases/name
- **Execution Control**: TestTime, RunFor, Priority, Confidential, Destructive, Environment, Releases, pattern, sets, trees, sources, arches, components, host_type_regex, hostRequires, kickstart, role, max_duration_seconds
- **Dependencies**: Requires, RepoRequires, RhtsRequires
- **Test Instructions**: run:, runtest:, build:, clean:, startup(), cleanup(), runtest()
- **Results Analysis**: waived

# CKI notes

- Not sure where to put trigger control:
  - pattern
    - sets, trees, sources, arches, components
- Note sure which of these are triggering and which are scheduling
  - triggering = when to run (what initiaties it)
  - scheduling = where to run? (what SUT and hardware to run it on)

# Elements - Jenkins

- actions
- description
- keepDependencies
- properties
- scm
- assignedNode
- canRoam
- disabled
- triggers
- concurrentBuild
- customWorkspace
- builders
- hudson.tasks.Shell
- builders
- publishers
- buildWrappers
- project
- blockBuildWhenDownstreamBuilding
- blockBuildWhenUpstreamBuilding

# Element categories - Jenkins

- **Info**: description, scm, project
- **Execution control**: actions, keepDependencies, properties, assignedNode, canRoam, disabled, trigger, concurrentBuild, customWorkspace, builders, publishers, buildWrappers blockBuildWhenDownstreamBuilding, blockBuildWhenUpstreamBuilding
- **Test Instructions**: Hudson.tasks.Shell

# Elements - SLAV

- repo
- yaml
- parser
- name
- timeout
- runTTL
- targets
- testOutFile
- device_type
- job_name
- timeouts (job/action)
- priority
- actions
- actions (deploy/boot/test)
  - deploy/images
    - uri
    - checksum_uri
    - checksum_type
    - compression
  - deploy/partition_layout
    - id
    - device_layout
    - size
    - type
  - boot (*)
    - login
    - password
    - prompts
    - failure_retry
    - timeout
    - input_sequence
    - wait_pattern
    - wait_time
  - test
    - failure_retry
    - name
    - test_cases
      - case_name
      - test_actions
        - boot (see *)
        - push
        - push/uri
        - push/dest
        - push/alias
        - run/name
        - pull/src
        - pull/alias

# Element categories - SLAV

- **Info**:
  - yaml, name, job_name, case_name
- **Execution control**:
  - timeout, repo, runTTL, targets, device type, timeouts/job, timeouts/action, priority, boot, login, password, prompts, failure_retry, input_sequence, wait_pattern, wait_time
- **Provisioning**:
  - deploy/images, uri, checksum_uri, checksum_type, compression, partition_layout, id, device_name, image_name, size, type
- **Instructions**:
  - boot, push, uri, dest, alias, run, name, pull, src, alias
- **Output parsing**:
  - pull?, src?, alias? parser, testOutFile

# Elements Comparison

- Information about a test (meta-data)
- Execution control
  - Pre-requisites and dependencies
  - Test variables
- Instructions for test execution
- Output parsing or conversion
- Results format
- Results analysis
- Report and notification data
- Visualization control

# Intersection - Information

- Common:
  - name, description, license, maintainer, version, package_version
  - source reference (repo, tarball, etc.)
- Uncategorized:
  - type, tags, data_files, arch, validpkgkeys, WEB_URL, suite, category, BUILT_FILES, PATH, Architecture
- Outliers:
  - Bug, PURPOSE, scm, author

# Intersection – Execution control

- Common:
  - timeout, <test variable>
  - source reference (tarball or git repo), source_sig
- Uncategorized:
  - (data for specifying allowed targets)
  - format, os, scope, device, environment, source, disk, fs, iosched, runtime, nr_task, disable_latency_status, nr_threads, cluster, priority, releases
  - trigger, concurrentBuild, canRoam, publishers, buildWrappers, blockBuild*
- Outliers:
  - reboot, rebuild, precleanup, postcleanup, specs
  - confidential, destructive, disabled

# Intersection - Dependencies

- Common:
  - kconfig, root, package, memory
- Uncategorized:
  - RepoRequires, RhtsRequires
- Outliers:
  - program, module, need_x, need_kernel_headers, need_cpu

# Pre-requisites

**Fuego:fuego_test.sh**

- Need variables:
  - NEED_MEMORY
  - NEED_FREE_STORAGE
  - NEED_KCONFIG
  - NEED_ROOT
  - NEED_PROGRAM
- Pre_check functions:
  - assert_define
  - is_on_target
  - is_on_sdk
  - assert_has_program

**CKI:Makefile**

- RunFor ?
- Requires
- RhtsRequires

# Pre-requisites as code

## Fuego:

### Functional.openct/fuego_test.sh

```
function test_pre_check {
    assert_has_program openct-control
    assert_has_program openct-tool
}
```

### Benchmark.hackbench/fuego_test.sh

```
NEED_ROOT=1

function test_pre_check {
    assert_define BENCHMARK_HACKBENCH_PARAMS
}
```

## CKI:

### perf/internal-testsuite/Makefile

```
@echo “RunFor:  perf” >>$METADATA
```

## Linaro:

```
check_root && \
    error_message “This script must be run as root”
```

# Intersection – Test instructions

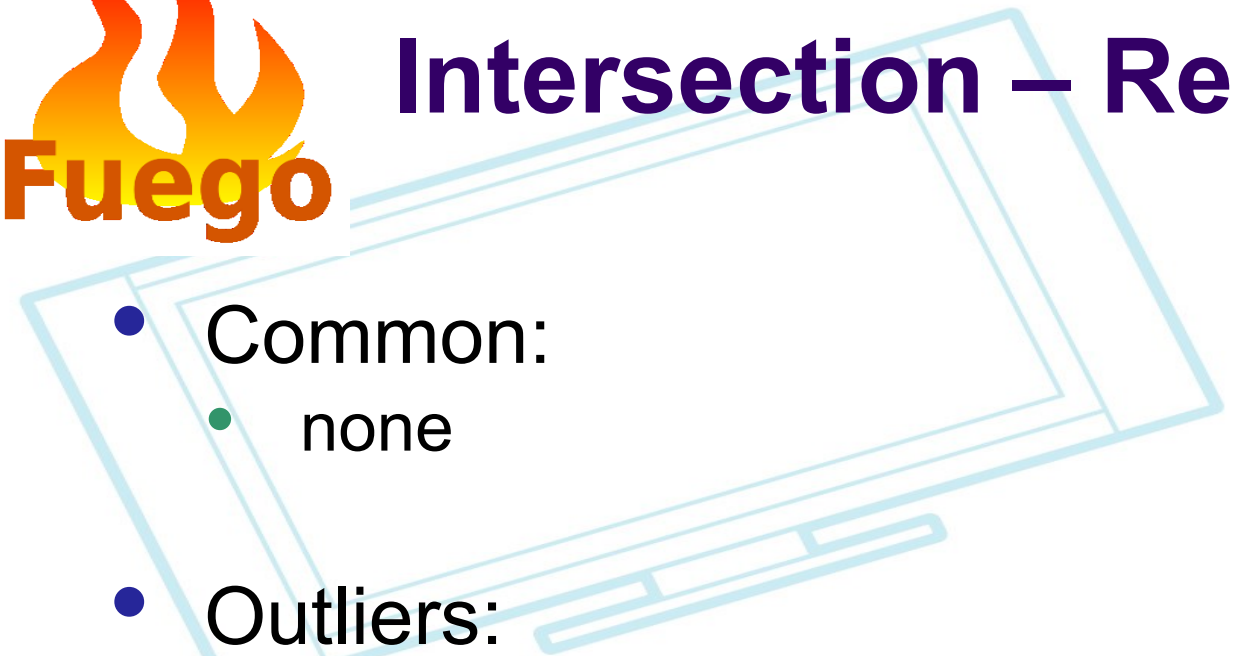
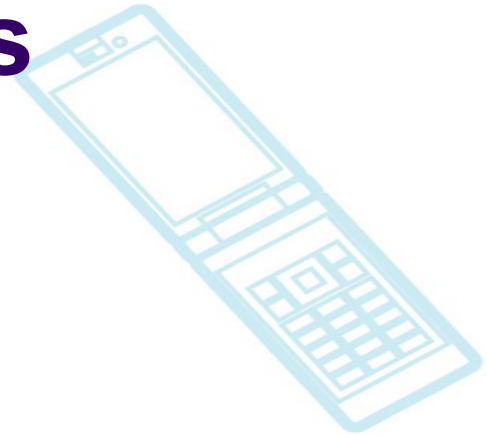
- Common:
  - cleanup, build, run, teardown
- Uncategorized:
  - pre_check(), deploy(), processing()
  - download(), package(), install()
- Outliers:
  - push, pull, src, dest, alias

# Intersection – Output parsing/conversion

- Common:
  - ???
- Uncategorized:
  - parse (python), parse (ruby), parse(awk), etc.
- Outliers:

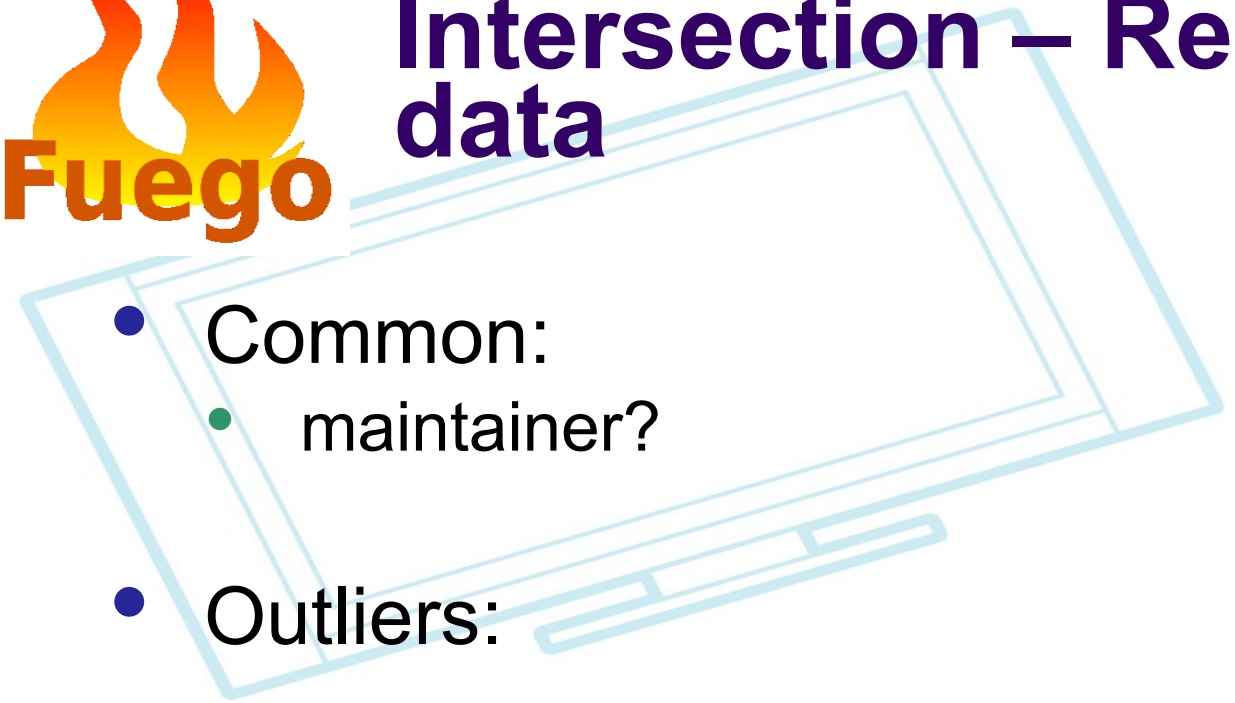
# Intersection – Results analysis

- Common:
  - none
- Outliers:
  - criteria.json, waive

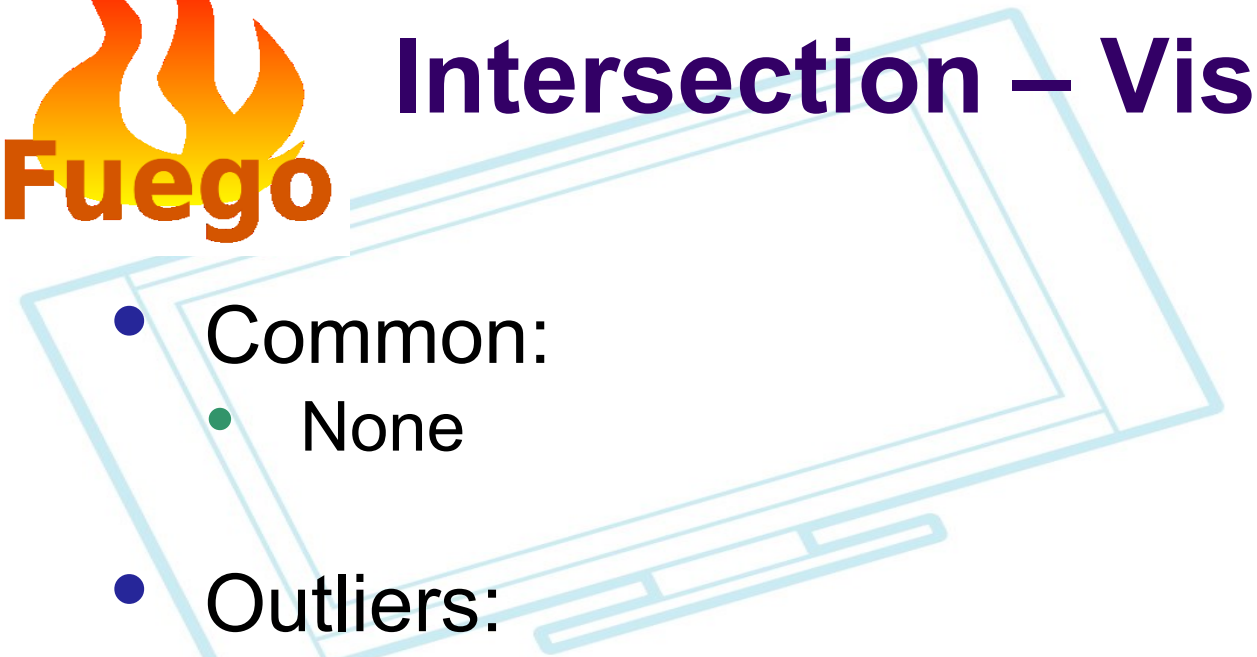
# Intersection – Report and notification data

- Common:
  - maintainer?
- Outliers:
- Status: not enough data was gathered




# Intersection – Visualization control

- Common:
  - None
- Outliers:
  - chart_config.json




# Results format

- Canonical format in which results are stored
  - e.g. database, kernelci json, xunit, etc.
- Parser = how it gets extracted from log
- Parser control variables:
  - Some systems have regular expression for log parsing
  - Some have dedicated code
- Major methods of handling test output differences:
  - All tests output the same – done by LTP (& kselftest w/ TAP?)
  - Convert on the fly – done by LAVA
  - Parse the log in post-processing – done by Fuego, 0day
  - ??? - CKI

# Results formats

- http://fuegotest.org/wiki/run.json
- https://api.kernelci.org/
- kcidb format

# Ideas for test definition harmony

- Add converter to/from a standard format
  - For each format
  - What do to about languages?
    - Conversions of declarative formats (json/yaml/markdown) should be easier
- Create a place to store a standard format
  - Is this really needed?
  - Can we just import another project's repository?
  - Do we need a test enumeration API?
- Try executing another framework's tests

# Converter issues

- Map our functions and elements onto the “standard” ones
  - Define same test phases (e.g. build, deploy, run, analyze)
  - Match pre-requisite and dependency names
  - Use common test variable (parameter) names and values
    - e.g. LOOPS for iozone
- Create a common library or toolset for common features
  - Fuego: subset of functions.sh, and fuego_board_function_lib.sh
  - Linaro: sh-test-lib
  - CKI: rhs-environment.sh, beakerlib.sh, rhts-make.include

# Harmonization issues

- Factoring of test definition elements
  - Time/location of execution
  - What files have each element
  - Concepts that one system has that the other doesn't
- Required software on target
- Required hardware in lab

# Place to share objects

- Project neutral site for collecting/disseminating objects
- or...
- Agreement to consolidate tests in one repository
- Possible uses:
  - Peer-to-peer test sharing
    - Eliminate gatekeeping for collaboration in testing community
  - Allow customization and enhancement of ad-hoc tests
    - For diagnosing problems
  - Apply tests to board that have hardware needed for test
    - Give access to developer who does not have hardware






# Let’s discuss the details....

# Thanks

Tim Bird

Fuego Test System Maintainer

Sr. Staff Software Engineer, Sony Electronics